APPENDIX A

1(a)----------

notes: In the Job class, Submit is the implementation of the client communications part through which a client requests the execution of the job. This implementation checks license and job validity to determine whether the batch job execution system is able to process the batch job [relevant to Claims 1, 9, 17]

source code excerpt:

```
    def Submit (self, op):
        # This routine must not be viewable or changeable by customers.  It
checks the license.
        # It must not call out to another module or the customer could
replace
        # that module.
        # If the license is not correct, it will raise basics.LicenseProblem
        Log("jdb", "Submission request received for job %s.", (self.ID(),))
        check_license()
        # Check validity
        if jobhelps.GetState(self) not in SubmitStates:
            Log("jdberr","Attempted submit for job %s in inappropriate
state.", (self.ID(),))
            raise basics.AccessDenied, "Submit"

        if op != job.JobOperation.Execute:
            Log("jdberr","Bad operation %d requested for job %s.", (op,
self.ID()))
            raise job.UnsupportedOperation
        self.SetAttributeNoSave(DatetimeAttribute ('SubmitDate',
time.time()))
        self.SaveToDisk("in job_impl.Submit")
        try:
            self.jobmanager().Run(self)
        except:
            raise job.UnsupportedOperation
```

APPENDIX A

1(b)----------

note: The ProcessSteps function extracts tasks (here called steps) from the batch job to be assigned to providers. [relevant to Claims 1, 9, 17, 24]

source code excerpt:

```
def ProcessSteps(job, stepList, stats, parent=None, save_to_disk=1):
    Log("jbex", "in ProcessSteps for %s", (str(job.ID()),))
    assnList = []

    if not parent:
        parent = job

    for step in stepList:
        step.dependentSteps = []

    # create dependencies for this stage's steps
    for step in stepList:
        provtype    = CachedGetAttribute(step, 'ProviderType').value[1]
        if provtype == jobhelps.NoopService:
            SetStatus(attrib=step, state=status.State.Done,
                secondary=status.SecondaryState.Success, dict=stats,
                save_to_disk=0)
            continue
            # no need to look for dependent steps, because nothing will
depend on a no-op
        step.parent = parent
        assn = InitializeStep(job=job, step=step, stats={},
            save_to_disk=0) # create empty values?
        assnValues = CachedGetAttribute(assn, 'Values').value[1]
        argattrib = CachedGetAttribute(step, 'Arguments').value[1]
        arguments = CachedGetAttributes(argattrib, [])
        for arg in arguments:
            argName   = arg.name
            argValue  = arg.value[1]
            argType   = CachedGetAttribute(argValue, 'Type').value[1]

            # Output Arguments - nothing to do unless it is a temp
destination
            if not arginfo.input_argp(provtype,argName) and argType !=
'temp':
                continue

            # Input Arguments
            # We only read argSource for cases where we need it (so temp
doesn't trip on it)
            if argType in ('job', 'param'):
                argSource = CachedGetAttribute(argValue, 'Source').value[1]
                jobVars = CachedGetAttribute(job, 'JobVariables').value[1]
                try:
                    valueAttribute = CachedGetAttribute(jobVars,
argSource).value
                except attributed.AttributeUnknown:
                    raise ProgramStructureError, "argument %s with type %s
has value %s and value has Source %s, which is not found in jobVars" %
(argName, str(argType), str(argValue), str(argSource))
```

```
            elif argType == 'super':
                argSource = CachedGetAttribute(argValue, 'Source').value[1]
                if not parent:
                    raise ProgramStructureError, "'super' can only be used in
an epilogue step"
                else:
                    values = parent.GetAttribute('Values').value[1]
                    try:
                        valueAttribute = values.GetAttribute(argSource).value
                    except:
                        raise ProgramStructureError, argSource
            elif argType == 'literal':
                argSource = CachedGetAttribute(argValue, 'Source').value[1]
                valueAttribute = StrAttVal(argSource)
            elif argType == 'temp':
                valueAttribute = StrAttVal(allocate_temp(job))
            elif argType == 'step':
                AddDependentStep(step, stepList, arg)
            else:
                raise ProgramStructureError, "Step: %s" %
CachedGetAttribute(step, 'Label').value[1]

            if argType != 'step':
                valueAttribute = attributed.Attribute(argName,
valueAttribute)
                CachedSetAttribute(obj=assnValues, attr=valueAttribute,
save_to_disk=0)

        # create assignment list
        if not step.argumentsNotReady: # this step is ready to be assigned
            assnList.append((job, step, assn))
            SetStatus(attrib=step, state=status.State.Processing,
                secondary=status.SecondaryState.None, dict=stats,
                save_to_disk=0)

    # create endSteps
    parent.endSteps = []
    for step in stepList:
        if not step.dependentSteps:
            parent.endSteps.append(step)

    if assnList:
        Log("jbex", "    - returning assignment list with %d assignments",
(len(assnList),))
        if save_to_disk: job.SaveToDisk("ProcessSteps with assnList")
        return assnList
    else:    # if there are no assignments for this stage, go to the next
stage
        result = NoAssignments(job=job, stats=stats, save_to_disk=0)
        if save_to_disk: job.SaveToDisk("ProcessSteps without assnList")
        return result
# end of ProcessSteps
```

APPENDIX A

1(c)----------

note: In the JMSAssigner class, AssignWork is the method which implements the assigning part: delegating a task to one of the plurality of service providers which gives a first signal, the first signal being the call to AssignWork. [relevant to Claims 1, 9, 13, 17, 24]

source code excerpt:

```
    def AssignWork(self, provid, manager, frequency):
        DebugLog("jmgr", '%s: request from %s', (repr(self), provid))

        # Hold lock while manipulating assigner state
        self.lock.acquire('Assigner.AssignWork')
        try:
            if self.asmtqueue:
                # Real assignment to hand out
                asmt = self.asmtqueue[0]
                del self.asmtqueue[0]
                self.pollers = []
                self.lastalloc = time.time()
                self.delaycount = 0
                if self.stat_dict:
                    self.stat_dict['queue-len'] = self.stat_dict['queue-
len']-1
            else:
                # Idle for now
                asmt = None
                if provid not in self.pollers:
                    self.pollers.append(provid)
                # Once we start handing out idle assignments it is
                # reasonable to expect providers to go away, so we
                # should be prepared to ask for a capacity increase when
                # load builds again in the future
                self.inhibitIncrease = 0
        finally:
            self.lock.release('Assigner.AssignWork')

        # Prepare assignment and record it
        # Note that asmt here is the assignment structure for the Provider
        # It is *NOT* the assignment attributed!
        if asmt:
            # Determine the actual reporting frequency for the Provider
            if frequency > asmt.reportfreq:
                asmt.reportfreq = frequency
            else:
                frequency = asmt.reportfreq

            # Record assignment on job
            self.jobmanager.AssignmentAllocated(asmt, provid, self)
            Log("jmgr", '%s: allocating %s to %s', (repr(self), asmt.id,
provid))
        else:
            self.idlecount = self.idlecount + 1
            asmt = self.MakeIdleAsmt('idle-%d' % self.idlecount)
        # Return
        return asmt
```